LIBRARY
TECHNICAL REPORT SECTION
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NPS52-81-015

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Measuring Control Structure Complexity
Through Execution Sequence Grammars

Bruce J. MacLennan

November 1981

Approved for public release; distribution unlimited

Prepared for:

Naval Postgraduate School
Monterey, CA 93940

FEDDOCS
D 208.14/2:NPS-52-81-015

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

D. A. Schrady
Acting Provost

The work reported herein was supported by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

Reproduction of all or part of this report is authorized.

This report was prepared by:

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-81-015	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Measuring Control Structure Complexity Through Execution Sequence Grammars		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Bruce J. MacLennan		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE November 1981 13. NUMBER OF PAGES 29
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

control structures, programming language metrics, complexity measures, software metrics.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A method for measuring the complexity of control structures is presented. It is based on the size of a grammar describing the possible execution sequences of the control structure. This method is applied to a number of control structures, including Pascal's control structures, Dijkstra's operators, and a structure recently proposed by Parnas. The verification of complexity measures is briefly discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

MEASURING CONTROL STRUCTURE COMPLEXITY THROUGH EXECUTION SEQUENCE GRAMMARS*

B. J. MacLennan

Naval Postgraduate School

Monterey, CA 93940

1. Abstract

A method for measuring the complexity of control structures is presented. It is based on the size of a grammar describing the possible execution sequences of the control structure. This method is applied to a number of control structures, including Pascal's control structures, Dijkstra's operators, and a structure recently proposed by Parnas. The verification of complexity measures is briefly discussed.

2. Introduction

Many questions face a language designer. Is a "while-do" better than a "repeat-until"? Is a "do-od" more complex than these? How does a "if-elsif-else" structure compare with nested "if-then-else"s? To this end it is useful to have a complexity measure for control structures that can serve as a *figure of merit* in making these determinations.

* The work reported herein was supported by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

In this paper we take the view that the complexity of a control structure is related to the complexity of the corresponding language of execution sequences. The complexity of this language can then be measured by determining the structural complexity of the corresponding grammar using techniques described in [4]. The motivation for this technique is the assumption that to understand a control structure a programmer must internalize the possible control sequences defined by that control structure. A further assumption is that the difficulty of doing this is approximated by the size of a grammar describing this class of execution sequences.

In the next section we informally present this technique by measuring the size of a conventional extended-BNF grammar for the language of execution sequences. The measurements depend on details of the concrete BNF notation that do not seem to be relevant to the control structure's complexity. Therefore, in the following section these measurement techniques are refined by measuring an abstract[1] grammar for the language; this eliminates irrelevant details of the concrete syntactic notation. Finally, we tabulate the complexity of a number of common control structures and discuss some limitations of the method.

1. By this we mean a grammar expressed in an abstract rather than a concrete form, not a grammar for an abstract language as opposed to a concrete language.

3. Concrete Grammar Size

3.1 Conditionals

We will begin our analysis with a simple control structure, the Pascal *if-then* statement. Consider an if-then such as this:

if B *then* S

and consider the possible execution sequences. These execution sequences can be written as regular expression, which use the operations catenation, union, Kleene cross, and Kleene star. Note that B will always be executed, but S will be executed only if B was true. Therefore the possible execution sequences are BS and B, depending on whether B was true or not (we represent consecutive execution by catenation). Hence, the set of possible execution sequences is

$$E\{\textit{if}\ B\ \textit{then}\ S\} = BS + B$$

where '+' represents the union of sets of execution sequences. If we then define the complexity C{X} of a construct to be the size of the execution sequence grammar of X,

$$C\{X\} = |E\{X\}|$$

then we can compute the complexity of the if-then. To measure the complexity of an execution sequence grammar we will take a very naive, concrete view, and count the tokens in the grammar. Thus,

$$C\{\underline{\text{if}}\ B\ \underline{\text{then}}\ S\} = |BS + B| = 4$$

since the tokens are 'B', 'S', '+', and 'B'.

Next we will consider the full if-then-else:

$$\underline{\text{if}}\ B\ \underline{\text{then}}\ S\ \underline{\text{else}}\ T$$

In this case it can be seen that B is always executed, followed by either S or T. Thus the possible execution sequences are BS and BT, which we can factor and write B(S + T). The assumption here is that the complexity is related to the shortest grammar for the language of execution sequences. Thus the complexity of the if-then-else is:

$$C\{\underline{\text{if}}\ B\ \underline{\text{then}}\ S\ \underline{\text{else}}\ T\} = |B(S+T)| = 6$$

It is a little more complex than the simple if-then, as is expected.

Finally, we will analyze the case-statement:

$$\underline{\text{case}}\ E\ \underline{\text{of}}\ (S_1;\ S_2;\ \ldots;\ S_n)$$

Clearly, E must be executed first, and then one of the S_i. The complexity is easy to calculate:

$$|E(S_1+S_2+\cdots+S_n)| = 2n+2$$

where n is the number of cases.

3.2 Iterative Constructs

Next we will analyze the Pascal repeat-until statement. Consider a repeat-until such as this:

repeat S until B

The effect of this is to execute S until B evaluates to true. Therefore, we execute S and then B. If B is false, we again execute S and B. This process continues until B becomes true (which must eventually happen in a terminating program). Therefore the possible execution sequences can be written

$$SB + SBSB + SBSBSB + \ldots$$

where concatentation denotes sequential execution and '+' can be read as "or". Really, the '+' denotes set union, since the above expression defines the set of possible execution sequences. Using exponential notation, the execution sequences for the repeat-until can be abbreviated:

$$SB + (SB)^2 + (SB)^3 + \ldots$$

Using the Kleene cross notation, this infinite union can be written

$$(SB)^+$$

and can be read one or more repetitions of the sequence SB. This agrees with the way we think of the behavior of a repeat-until. The complexity of this construct is measured simply:

$$C\{\underline{\text{repeat}}\ S\ \underline{\text{until}}\ B\} = |(SB)^+| = 5$$

Very much the same analysis can be applied to Pascal's while-do:

$$\underline{\text{while}}\ B\ \underline{\text{do}}\ S$$

This construct executes B; if the result is false it terminates, otherwise it executes S and loops back to test B again. Therefore we can write the execution sequences

$$B + BSB + BSBSB + BSBSBSB + \ldots$$

That is,

$$B + B(SB)^1 + B(SB)^2 + B(SB)^3 + \ldots$$

Now, if we use ϵ to represent the null execution sequence, then B can be factored out of the above expression:

$$B[\ \epsilon + (SB)^1 + (SB)^2 + (SB)^3 + \ldots\ ]$$

This can be simplified with Kleene's cross:

$$B[\ \epsilon + (SB)^+\ ]$$

It now becomes apparent that this can be simplified even further by using the Kleene star notation, since

$$C^* = \epsilon + C^+$$

Thus, we can compute the complexity of the while-do:

$$C\{\underline{\text{while}}\ B\ \underline{\text{do}}\ S\} = |B(SB)^*| = 6$$

Notice that the while-do is slightly more complex than the repeat-until because of the leading inital test of the condition. It would probably be more intuitive to ignore the final failure test of B and analyze the while-do as:

$$C\{\underline{\text{while}}\ B\ \underline{\text{do}}\ S\} = |(BS)^*| = 5$$

which agrees with our intuitive notion that a while-do and repeat-until have about the same complexity. It is not known at this time which measurement technique is correct.

Since we have considered leading-decision loops and trailing-decision loops, we will next analyze mid-decision loops. A mid-decision loop has the form:

$$\underline{\text{loop}}\ S\ \underline{\text{exit}}\ \underline{\text{when}}\ B;\ T\ \underline{\text{end}}\ \underline{\text{loop}}$$

The meaning of this is: execute S, then test B, if B is true terminate the iteration, otherwise execute T and continue looping. Mid-decision loops are often useful in search operations. It is easy to see that the execution sequences are:

$$SB + SBTSB + SBTSBTSB + \ldots$$

or in general,

$$SB(TSB)^*$$

The resulting complexity is

$$C\{\underline{\text{loop}}\ S\ \underline{\text{exit}}\ \underline{\text{when}}\ B;\ T\ \underline{\text{end}}\ \underline{\text{loop}}\} = |SB(TSB)^*| = 8$$

As would be expected, this is more complex than either the leading or trailing decision loops.

So far we have described execution sequences using just the operators used in regular expressions (viz., Kleene cross, Kleene star, union, and catenation). Recently, however, several extended BNF notations (see, for example, [2, 3, 5, 8]) have adopted an operator that expresses a very common configuration, the delimited sequence. An example is "a sequence of names separated by commas." Using the regular expression operators, this would have to be written

<name> (, <name>)*

which requires repeating <name>. The delimited sequence notation allows this to be expressed directly:

<name> , ...

In general, 'CD...' means the class of all non-empty sequences of Cs alternating with Ds; that is, C(DC)*. Using this notation, which expresses a very simple structural idea, the leading-decision and mid-decision loops have the complexity:

C{while B do S} = |BS...| = 3

C{loop S exit when B; T end loop} = |(SB)T...| = 5

This may seem like an ad hoc definition of an operator to simplify the description of these execution sequences. For this reason we have restricted our attention to notations that have

already proved useful in describing sets of sequences. As we have said, the delimited sequence operator has been independently proposed by several authors as embodying a useful configuration. Whether it should be used in measuring control structure complexity remains an open question.

3.3 Dijkstra's Constructs

In this section we analyze Dijkstra's if-fi and do-od control structures [1]. The if-fi has this form:

$$\underline{\text{if}}\ B_1 \rightarrow S_1 \ \square \ \cdots \ \square \ B_n \rightarrow S_n\ \underline{\text{fi}}$$

The guards B_1, ..., B_n are evaluated non-deterministically. If one or more evaluates to true, then one of the corresponding statements S_i is chosen and executed. If none of the guards is true, then an error condition exists and the program aborts. Thus, the possible execution sequences are:

$$B_1S_1 + B_2S_2 + \cdots + B_nS_n$$

The size of this expression is 3n-1, so the complexity of the if-fi is:

$$C\{\underline{\text{if}} \cdots\ B_i \rightarrow S_i\ \ldots \underline{\text{fi}}\} \ = \ 3n-1$$

The do-od is an iterative construct patterned on the if-fi. It has the form:

$$\underline{\text{do}}\ B_1 \rightarrow S_1 \ \square \ \cdots \ \square \ B_n \rightarrow S_n\ \underline{\text{od}}$$

On each iteration the guards are evaluated non-deterministically.

If none of them are true, the loop terminates. Otherwise one of the corresponding S_i is selected, and the loop repeats. It is easy to see that the execution sequences are

$$(B_1S_1 + \cdots + B_nS_n)^*$$

Therefore the complexity of the do-od is:

$$C\{\underline{do} \cdots B_i \rightarrow S_i \cdots \underline{od}\} = 3n+2$$

which is approximately the same as the if-fi.

It is instructive to compare the complexity of the if-fi (3n-1) with that of the more conventional if-elsif-else (or multi-branch conditional). Effectively, we are comparing the complexities of non-deterministic and deterministic conditionals. The if-elsif-else has the form:

$$\underline{if}\ B_1\ \underline{then}\ S_1\ \underline{elsif}\ B_2\ \underline{then}\ S_2 \cdots \underline{else}\ E\ \underline{endif}$$

This is executed strictly sequentially; if B_1 is false, then B_2 is tried, if B_2 is false, then B_3 is tried, and so forth. This is equivalent to nested if-then-else statements. It is easy to write down the execution sequences:

$$B_1S_1 + B_1B_2S_2 + \cdots + B_1B_2\cdots B_nE$$

The length of this regular expression is

$$2 + 3 + \ldots + (n+1) = \sum_{i=1}^{n} (i+1) = \frac{n^2+3n}{2}$$

This is not the complexity, however, since this regular

expression can be simplified by factoring to

$$B_1(S_1 + B_2(S_2 + \cdots B_n(S_n + E) \cdots))$$

We can find the length of this expression inductively. Let

$$E_i = B_i(S_i+E_{i+1})$$

for $i \leq n$, and $E_{n+1} = E$. Then,

$$|E_i| = 5 + |E_{i+1}|$$

and $|E_{n+1}| = 1$. Therefore $|E_1| = 5n+1$. In summary, the complexity of the n-branch if-elsif-else is

$$C\{\underline{\text{if}} \cdots \underline{\text{elsif}}\ B_i\ \underline{\text{then}}\ S_i \cdots \underline{\text{else}}\ E\ \underline{\text{endif}}\} = 5n+1$$

Thus, the complexity of the non-deterministic if-fi, 3n-1, is considerably less than that of the deterministic if-elsif-else, 5n+1.

4. Abstract Grammar Size

4.1 Introduction

The reader will have probably noticed that our complexity measurements include aspects of the regular expression notation, such as parentheses, that on an intuitive basis are not very relevant. Previous work [4, 6] has shown that better measurements are obtained if an abstract form of the grammar is measured, rather than some concrete representation, such as we have used in the first section. This approach will count operators

that alter the sets of execution sequences, such as '+', '*', and catenation, while ignoring those that do nothing, such as parentheses. Previous work has also shown that it is best to count multi-armed alternations as a single operator, rather than several. That is, an expression such as

$$S_1 + S_2 + \cdots + S_n$$

(which would normally be counted as 2n-1) will be analyzed as though it were written

$$\Sigma[S_1, S_2, \ldots, S_n]$$

which gives it a count of n+1 (n for the S_i and 1 for the Σ). Since we are counting the operators that "do something" we will now have to also count catenation, so we will write ST explicitly as S·T

4.2 Recomputation of Complexities

In this section we will recompute the complexities of the constructs analyzed using concrete grammars. The Pascal control structures are trivial:

C{if B then S}	|B·S+B|	5
C{if B then S else T}	|B·(S+T)|	5
C{while B do S}	|(B·S)*|	4
C{repeat S until B}	|(S·B)+|	4
C{loop S exitwhen B; T end loop}	|(S·B)T...|	5
C{case E (...S_i...)}	|E·Σ[...S_i...]|	n+3

The results of these measurements are not very different from those based on the concrete grammar.

Next we will consider Dijkstra's constructs, which will make use of the $\bar{\Sigma}$ operator. The if-fi is analyzed

$$C\{\underline{\text{if}} \cdots B_i \rightarrow S_i \cdots \underline{\text{fi}}\} = |\bar{\Sigma}[\ \ldots, B_i \cdot S_i, \cdots\ ]| = 3n+1$$

The result is almost the same as with the concrete grammar; the addition of the catenation operations has compensated for the omitted unions (+).

The do-od construct is exactly analogous:

$$C\{\underline{\text{do}} \cdots B_i \rightarrow S_i \cdots \underline{\text{od}}\} = |\bar{\Sigma}[\ \ldots, B_i \cdot S_i, \cdots\ ]^*| = 3n+2$$

The execution sequences of the if-elsif-else are:

$$B_1 \cdot (S_1 + B_2 \cdot (S_2 + \cdots B_n \cdot (S_n + A) \ \ldots))$$

In this case the inductive equation is

$$E_i = B_i \cdot (S_i + E_{i+1})$$

Therefore each clause adds 4, resulting in a total complexity:

$$C\{\text{if-elsif-else}\} = 4n+1$$

This is a significantly lower measurement than that obtained with the concrete grammar (5n+1), largely owing to the abstract grammar's insensitivity to parentheses.

5. The Parnas It-Ti Construct

5.1 The Non-Deterministic It-Ti

In this section we analyze the complexity of a new control structure proposed by Parnas [7]. This control structure is a combination of Dijkstra's if-fi and do-od structures and has the form:

$$\underline{\text{it}}\ B_1 \rightarrow S_1 X_1\ V \cdots V\ B_n \rightarrow S_n X_n\ \underline{\text{ti}}$$

The X_i'is either an up-arrow indicating continuation of the iteration or a down-arrow indicating termination of the iteration. The semantics of the it-ti is as follows: The guards are evaluated non-deterministically. Out of the ones that evaluate to true, one is chosen and its corresponding statement S_i is executed. When this statement has completed the continuation X_i is considered. If it is repeat (an up-arrow) then the it-ti loops again; if it is break (a down-arrow) then the it-ti terminates.

Since the it-ti described above is non-deterministic, the order of its arms can be changed without altering its meaning. This simplifies the analysis of the it-ti because the repeating arms and the breaking arms can be grouped together. We will assume that there are m repeating arms, and that they are moved to the front of the it-ti. The complexity is then easy to calculate:

$$C\{\underline{it}\ B_1 \rightarrow S_1 \text{repeat } V \cdots V\ B_m \rightarrow S_m \text{repeat}$$
$$V\ B_{m+1} \rightarrow S_{m+1} \text{break } V \cdots V\ B_n \rightarrow S_n \text{break } \underline{ti}\}$$
$$= |\bar{\Sigma}[B_1 \cdot S_1, \ldots, B_m \cdot S_m]^* \cdot \bar{\Sigma}[B_{m+1} \cdot S_{m+1}, \ldots, B_n \cdot S_n]|$$
$$= 3m+2 + 3(n-m)+1$$
$$= 3n + 3$$

Thus the complexity is comparable to that of the if-fi and do-od.

5.2 The Deterministic It-Ti

In this section we analyze a variant of the it-ti defined by Parnas called the deterministic it-ti. This has the form

$$\underline{it}\ B_1 \rightarrow S_1 X_1\ \underline{else}\ \underline{or} \cdots \underline{else}\ \underline{or}\ B_n \rightarrow S_n X_n\ \underline{ti}$$

In this construct the guards are executed strictly sequentially. In other words, if B_1 is true, then S_1 is executed and continuation action X_1 is taken; otherwise testing continues with B_2. As for the non-deterministic it-ti, an error condition exists if none of the guards is true.

The analysis of the deterministic it-ti is considerably more complicated than the non-deterministic since the arms cannot be rearranged to group the repeating and breaking arms together. In fact, each different arrangement of breaks and repeats (i.e., of the X_i) effectively defines a different control structure. To keep the mathematics tractable we introduce several abbreviations. The notation

$$E\langle x_1 x_2 \cdots x_n \rangle$$

represents the execution sequences of a deterministic it-ti whose i-th continuation action is x_i. We will use 'b' to represent 'break', 'r' to represent 'repeat', 'x' to represent either 'b' or 'r', 'X' to represent a sequence of either 'b's or 'r's, and 'B' to represent a sequence of 'b's. These notations will just be used inside the angle brackets of E<...>.

We will also make one change to the semantics of the deterministic it-ti to simplify the analysis. If none of the guards are true, we will assume that the it-ti "falls through" like a do-od. Later we will correct the formula to account for the fact that this is an error condition in Parnas' formulation.

The formula will be derived by an inductive process starting with the degenerate it-ti that contains no arms, viz., it ti. This is a fall-through, and the corresponding execution sequence is the null sequence, so

$$E\langle\rangle = \epsilon$$

We will next investigate extensions of an it-ti formed by adding a new arm to the beginning. The formulas for the execution sequences are derived by a variant of the method of undetermined coefficients suggested by R.W. Hamming. In this method, the general form of a formula is assumed and its specific coefficients or parts are derived. Deterministic it-ti's are of two sorts: those that contain only 'break's (and are hence multi-branch conditionals), and those which contain at least one 'repeat'. The latter we will assume have an execution sequence of the form

$L^{*}\cdot T + U$, for some regular expressions L, T, and U. First, however, we will address it-tis with only 'break's.

Consider an it-ti of the form bB, i.e., all of whose arms are 'break's. We want to calculate the execution sequences E<bB>. Suppose the arm corresponding to the b is C→S, then the possible execution sequences of bB are C·S or C·E<B>, where E<B> is the set of execution sequences of the reduced it-ti B. This can be factored giving,

$$E<bB> = C\cdot(S + E<B>)$$

Next consider an it-ti of the form rB, i.e., a repeating arm followed by all breaking arms. Suppose the repeating arm is C→S. Then, if C is true, S will be executed and the it-ti will repeat. Otherwise the it-ti B is executed (which is just a multi-branch conditional). Therefore the possible execution sequences are

$$E<rB> = (C\cdot S)^{*}\cdot C\cdot E<B>$$

Next we will consider extensions of an it-ti containing at least one repeat, X. Thus we will derive E<xX> from E<X>. By the method of undetermined coefficients, we will assume E<X> to have the form $L^{*}\cdot T + U$ (since by assumption it contains a loop).

Consider first the case of adding a breaking arm C→S; we wish to calculate E<bX>. The effect of this it-ti is to evaluate C; if it's true then evaluate S and break; otherwise continue with the execution of X. Therefore the execution sequences are

$$E\langle bX\rangle = (C\cdot L)^{*}\cdot T + C\cdot(S+U)$$

This can be seen to have the form $L^{*}\cdot T+U$.

Next we will consider the case of adding a repeating arm C→S; we wish to calculate E<rX>. The effect of this it-ti is to evaluate C; if it's true then evaluate S and repeat; otherwise continue with the execution of X. Therefore the execution sequences are

$$E\langle rX\rangle = [C\cdot(S+L)]^{*}\cdot T + C\cdot U$$

Again, this has the form $L^{*}\cdot T+U$, so our use of the method of undetermined coefficients has been successful.

It is now a routine matter to calculate the complexity of these regular expressions.

$$|E\langle\rangle| = 0$$

$$|E\langle bB\rangle| = |E\langle B\rangle| + 4$$

$$|E\langle rB\rangle| = |E\langle B\rangle| + 7$$

$$|E\langle bX\rangle| = |L| + |T| + |U| + 9 = |E\langle X\rangle| + 6$$

$$|E\langle rX\rangle| = |L| + |T| + |U| + 9 = |E\langle X\rangle| + 6$$

The last two equations follow from the fact that

$$|E\langle X\rangle| = |L| + |T| + |U| + 3$$

since $E\langle X\rangle = L^{*}\cdot T + U$.

These equations can now be solved for the complexity. Consider first the case of an it-ti all of whose arms are breaks,

$C\{B\} = |E\langle B\rangle|$. You can see from the equations above that each arm adds 4 to the complexity. Therefore, if there are m breaks $b_1 b_2 \cdots b_m$, the complexity is

$$C\{b_1 \cdots b_m\} = 4m$$

Next consider an it-ti with one repeat followed by m breaks. This has the form $rb_1 \ldots b_m$. The complexity is

$$C\{rb_1 \cdots b_m\} = C\{b_1 \ldots b_m\}+7 = 4m+7$$

Finally, we have the case of adding either a break or a repeat to an it-ti that already contains a mixture of breaks and repeats. Regardless of whether the new arm is a break or repeat, it adds 6 to the complexity. Therefore, if k arms are added the complexity is increased by 6k:

$$C\{x_1 \ldots x_k r b_1 \ldots b_m\} = 6k + 4m + 7$$

It is already apparent that the deterministic it-ti is a complex control structure since there is a factor of 6 involved.

To be able to compare the it-ti with other control structures it is useful to have its complexity in terms of n, the number of arms. Note that $n = k+m+1$ if there is at least one repeat, otherwise $n = m$. Therefore if there are no repeats we have

$$C\{B\} = 4n$$

If there is at least one repeat we have

$$C\{X\} = 6k+4m+7 = 4(k+m+1) + 2k + 3 = 4n + 2k + 3$$

This is still not a very convenient form, since k is one less than the number of arms that aren't terminal breaks, a rather unintuitive quantity. It is more convenient to express the complexity either in terms of m, the number of terminal breaks, or in terms of s=n-m, the number of arms that aren't terminal breaks:

$$C\{X\} = 6n-2m+1 = 4n+2s+1$$

Since m can vary from n to 0 it's easy to see that the complexity of the deterministic it-ti can vary from 4n to 6n+1. All of these are considerably more complex than the non-deterministic it-ti's 3n+3.

To account for the fact that Parnas' it-ti aborts if none of the guards are satisfied, it is merely necessary to add an additional breaking arm to the end of the form 'true→abort break'. This increases the complexity to 6n-2m+5 (or 4n+2s+5).

Notice that the deterministic it-ti is the first construct we have encountered whose complexity depends on another parameter besides n, the number of arms. This reflects the fact that the deterministic it-ti is in fact a <u>family</u> of control structures, since each different arrangement of repeats and breaks defines a different pattern of control flow.

6. Conclusions

The complexities calculated for the various control structures are summarized in the following table. This table also show the complexity per arm to facilitate comparisons between structures with a fixed number of arms (e.g., the if-then-else) and those with a variable number of arms.

control structure	complexity	per arm
if-then	5	5
if-then-else	5	2.5
while-do	4	4
repeat-until	4	4
mid-decision loop	5	2.5
case	n+3	1+
multi-branch if	4n+1	4+
if-fi	3n-1	3-
do-od	3n+2	3+
non-deterministic it-ti	3n+3	3+
deterministic it-ti	6n-2m+5	4 - 6

Figure 1. Control Structure Complexities

The complexities are also shown graphically in the following figure. These measures seem to agree with our intuitive estimations of the relative complexity of these control structures.

Whenever a measure such as this is proposed the question of its validation must be asked. In other words, is this the correct complexity measure? Complexity is used in many senses. Perhaps the most common uses relate to the difficulty of understanding. That is, one thing is more complex than another if it is more difficult to understand. The implication seems to be that complexity is a psychological property that requires psychological techniques in its verification. However, this is not the case.

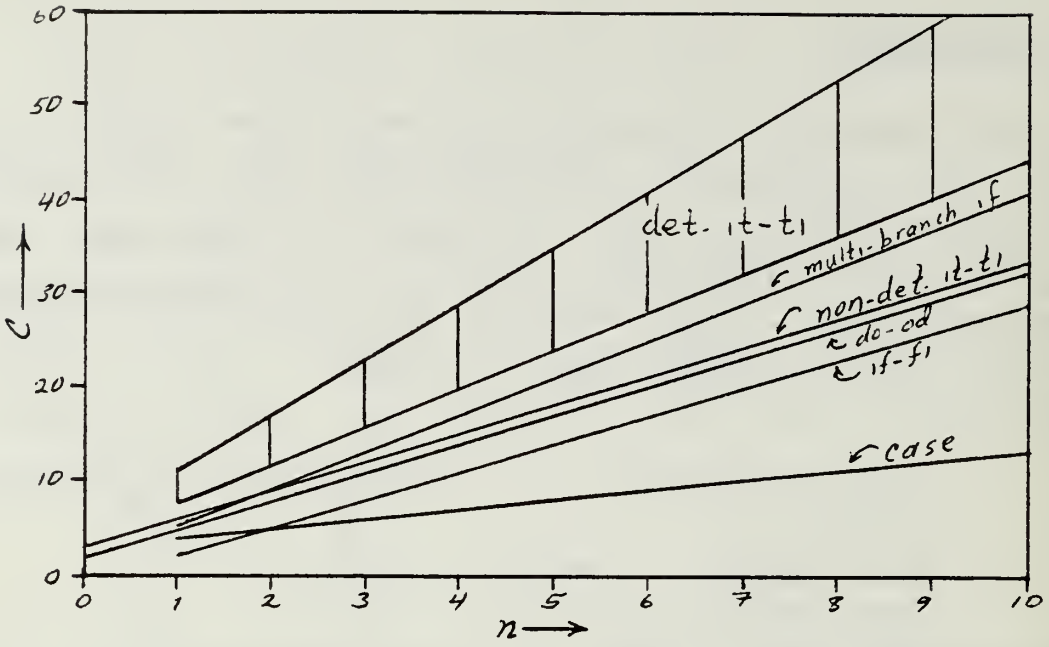

Figure 2. Complexities of Various Control Structures

An analogy may help to clarify the issues. When we hold an object in our hands we experience a psychological property, a sensation of weight. This property depends on many circumstances, including the shape of the object, how long it's held, and so forth. Similarly, our sensation of time can be quite subjective and can depend on many circumstances. Psychological weight and psychological duration are valid objects of scientific inquiry and in fact have been studied by psychologists. These properties are analogous to psychological complexity, the perceived complexity of a system.

Although our first notions of time were based on psychological duration and our first notions of weight on psychological weight, these are not the only notions of time and weight that we now use. Physicists have discovered notions of time and weight that are objective, i.e., that are independent of individual

psychologies. Time is measured by clocks even though we realize that there is often only a loose correlation between clock time and psychological time. Similarly, the concepts weight and mass are defined and measured in completely non-psychological terms. The measurement of physical duration and physical weight is a problem of physics; the measurement of psychological duration or weight is a problem of psychology, as is the establishment of the relation between the physical and psychological properties.

Physicists have studied the physical properties rather than the psychological properties because they have found the physical properties to be more easily reproduced in experiments. That they can be measured objectively is certainly significant, since it eliminates a dependence on a very imperfectly understood entity, human psychology. Even more importantly however, the physical properties have been found to be part of a highly integrated system of laws and principles that have been very productive in understanding the world. In other words, these physical properties have great practical value.

How does this apply to complexity measures? We can of course try to understand the phenomenon of psychological complexity; this is a fruitful area of research for psychologists. Our analogy suggests, however, that there is another useful notion of complexity, that there may be a non-psychological measure of complexity. This paper has presented one such measure. Whether it turns out to be the "right" measure or not will depend largely on whether it can be integrated into a comprehensive, practical

theory. Such an integration should also resolve some of the measurement ambiguities, such as how the delimited sequence operator should be counted in measurements. In the meantime it must remain as one possible notion of complexity. We should not be suprised at this state of affairs; it took many years for physicists to settle on definitions of work, force, mass, etc.

7. References

[1] Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, 1976.

[2] MacLennan, B.J., Semantic and Syntactic Specification and Extension of Languages, Purdue University PhD Dissertation, December 1975.

[3] MacLennan, B.J., Prototype Linear Argot System Users' Manual, June 1978, available from author.

[4] MacLennan, B.J., The Structural Analysis of Programming Languages, Naval Postgraduate School Computer Science Department Technical Report NPS52-81-009, September 1981.

[5] MacLennan, B.J., The Automatic Generation of Syntax Directed Editors, Naval Postgraduate School Computer Science Department Technical Report NPS52-81-014, October 1981.

[6] MacLennan, B.J., An Investigation of System Complexity, Naval Postgraduate School Computer Science Department

Technical Report NPS52-81-010.

[7] Parnas, D.L. An Alternative Control Structure and Its Formal Definition, unpublished, 1981.

[8] Wulf, W.A., et al. (Preliminary) An Informal Definition of Alphard, Carnegie-Mellon University Computer Science Department technical report, November 29, 1977.

INITIAL DISTRIBUTION LIST

	No. copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, CA 93940	40
Professor Bruce J. MacLennan, Code 52Ml Department of Computer Science Naval Postgraduate School Monterey, CA 93940	12

U199438

DUDLEY KNOX LIBRARY - RESEARCH REPORTS
5 6853 01071266 4
U199438